MVA **Microsoft Virtual Academy**
Formation gratuite en ligne

Ne faites jamais confiance aux données reçues : la faille XSS

Par Mathieu Nebra alias Mateo21

Difficulté
Facile

Note

Thématiques
Base de données, MySQL, PHP, Web

Mis à jour le vendredi 3 mai 2013

Les éléments du formulaire L'envoi de fichiers

Vous vous souvenez des mises en garde que j'avais faites dans le chapitre précédent ? Elles ne concernaient pas que les paramètres qui transitent par l'URL : tout cela vaut aussi pour les formulaires !

Mais… autant je vois comment on peut modifier l'URL, autant je ne comprends pas comment peut faire un visiteur pour modifier le formulaire de mon site et trafiquer les données !

On a tendance à croire que les visiteurs ne peuvent pas « bidouiller » le formulaire mais c'est faux. Je vais dans un premier temps vous montrer pourquoi les formulaires ne sont pas plus sûrs que les URL, puis je vous parlerai d'un autre danger important : la faille XSS. Avec ça, nous aurons vraiment fait le tour de ce qu'il faut savoir pour être tranquilles par la suite !

Pourquoi les formulaires ne sont pas sûrs

Tout ce que nous avons appris dans le chapitre précédent sur les URL reste valable ici. Toutes les informations qui proviennent de l'utilisateur, à savoir les données de `$_GET` et de `$_POST`, doivent être traitées avec la plus grande méfiance.

Vous ne pouvez pas supposer que vous allez recevoir ce que vous attendiez.

Cette règle est très simple, mais je vous propose un exemple concret pour bien visualiser le problème. Imaginez que vous demandez à vos visiteurs de rentrer dans un champ leur date de naissance *au format JJ/MM/AAAA*. Combien vont respecter cette mise en forme ? Combien vont se tromper par erreur ? Je vous garantis que parmi tous vos visiteurs, alors que vous attendiez quelque chose comme « 04/10/1987 », vous allez tomber sur une personne qui va écrire : « Je suis né le 4 octobre 1987 ». C'est un exemple un peu extrême mais ça va vous arriver, soyez-en sûrs. Par conséquent, quand vous ferez le traitement de la date en PHP, il faudra bien vérifier qu'elle respecte le format que vous avez indiqué.

Pour vérifier si une chaîne de texte correspond à un certain format, comme « JJ/MM/AAAA », on peut utiliser une validation par *expression régulière*. Les expressions régulières feront l'objet de deux chapitres vers la fin de ce livre car il s'agit d'un sujet assez complexe.

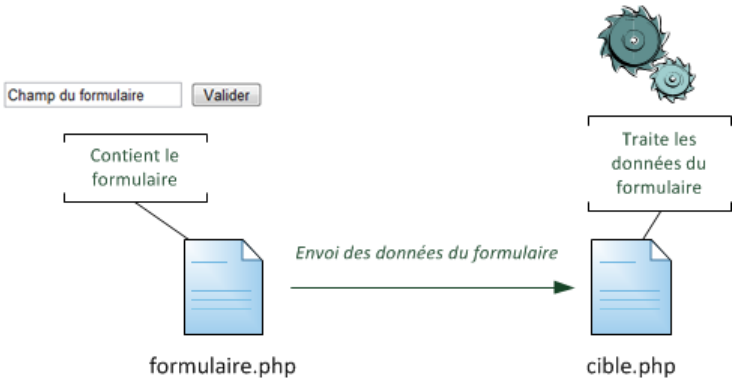
De la même manière, comme dans le chapitre précédent, même si vous demandez un nombre compris entre 1 et 100, il y aura bien quelqu'un pour écrire « 48451254523 ». Soyez donc vigilants et n'ayez jamais confiance en ce qui vient de l'utilisateur, à savoir les données issues des arrays `$_GET` et `$_POST`.

Avec les formulaires, vous ne pouvez pas non plus supposer qu'on va vous envoyer tous les champs que vous attendiez. Un visiteur peut très bien s'amuser à supprimer un champ de texte, et dans ce cas votre page `cible.php` ne recevra jamais le texte qu'elle attendait ! Il faudra impérativement qu'elle vérifie que toutes les données qu'elle attendait sont bien là avant d'effectuer la moindre opération.

Puisque la page du formulaire se trouve sur mon site, comment peut faire un visiteur pour modifier ma page web ? Il peut voir les sources mais pas les modifier !

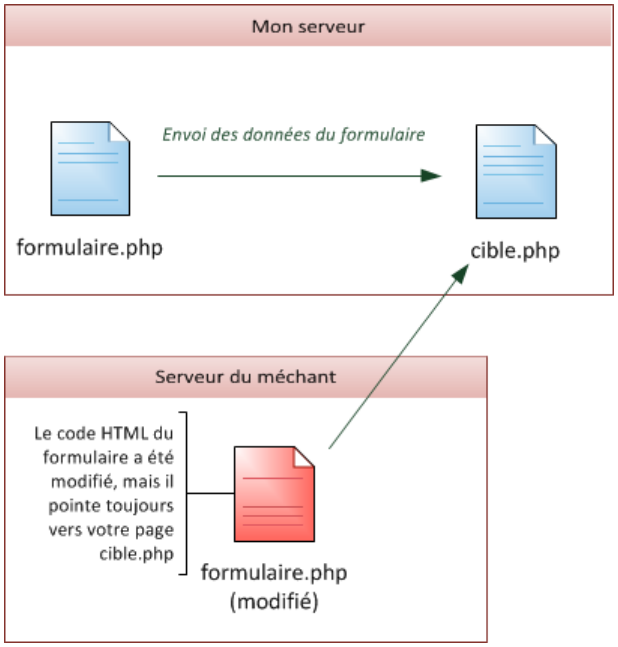
En effet, vos visiteurs ne peuvent pas modifier vos pages web sur le serveur… Mais ils peuvent les reprendre et les modifier ailleurs.

Souvenez-vous du schéma de la figure suivante :



La page `formulaire.php` contient le formulaire et `cible.php` traite les données qu'on lui a envoyées. Autant le code PHP n'est jamais visible par vos visiteurs, autant le code HTML du formulaire, lui, peut être vu par tout le monde.

À partir de là, qu'est-ce qui empêche quelqu'un de créer une copie légèrement modifiée de votre formulaire et de la stocker sur son serveur, à l'image de la figure suivante ?



Le formulaire modifié

Sur le schéma de la figure suivante, le « méchant » (nous l'appellerons comme ça, parce que ça lui va bien. ;-)) a pris le code HTML de votre formulaire, l'a modifié et l'a enregistré sur son serveur (ou même sur son ordinateur). L'attribut `action` a été modifié pour indiquer l'adresse absolue (donc complète) de votre page cible :

html+php
```
<form method="post" action="http://www.monsite.com/cible.php">
```

Le méchant peut maintenant modifier votre formulaire, ajouter des champs, en supprimer, bref faire ce qu'il veut avec ! Votre page `cible.php` n'y verra que du feu car il est **impossible** de savoir avec certitude de quel formulaire vient le visiteur.

Ces explications sont assez techniques. En fait, on les réserve normalement aux personnes plus expérimentées que les débutants. Cependant, je tenais volontairement à vous montrer comment c'est possible. Même si tout n'est pas totalement clair dans votre tête, vous avez au moins **l'idée** du mode de fonctionnement.

S'il y a une chose à retenir ici, c'est que **les formulaires sont modifiables par tous les visiteurs** contrairement à ce qu'on pourrait penser. Par conséquent, votre page `cible.php` devra être aussi vigilante que nous l'avons été dans le chapitre précédent et ne pas faire confiance aux données de l'utilisateur (les programmeurs ont d'ailleurs une maxime : « **Never trust user input** », ce qui signifie « Ne faites jamais confiance aux données de l'utilisateur »).

Il y a un moyen encore plus simple de modifier le formulaire de votre site sans avoir accès à votre serveur. Internet Explorer 8 et Google Chrome embarquent des « outils pour les développeurs » qui permettent de modifier le code HTML de la page que l'on visite en temps réel. Firefox peut faire de même avec son célèbre plugin Firebug.

La faille XSS : attention au code HTML que vous recevez !

Il nous reste un dernier point important à voir ensemble et après, j'arrête de vous faire peur, promis !

La faille XSS (pour **cross-site scripting**) est vieille comme le monde (euh, comme le Web) et on la trouve encore sur de nombreux sites web, même professionnels ! C'est une technique qui consiste à injecter du code HTML contenant du JavaScript dans vos pages pour le faire exécuter à vos visiteurs.

Reprenons la page qui affiche le prénom qu'on lui envoie. Elle contient notamment le code suivant :

html+php
```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <?php echo $_POST['prenom']; ?> !</p>
```

Si le visiteur décide d'écrire du code HTML à la place de son prénom, cela fonctionnera très bien ! Par exemple, imaginons qu'il écrive dans le champ

« Prénom » le code : `<strong>Badaboum</strong>`. Le code source HTML qui sera généré par PHP sera le suivant :

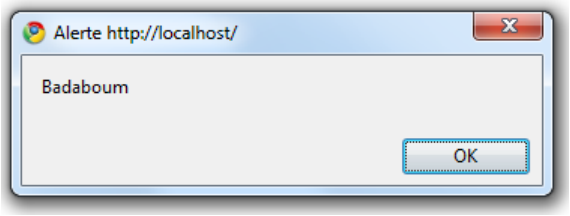
html+php
```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <strong>Badaboum</strong> !</p>
```

Et alors ? S'il veut mettre son prénom en gras, c'est son problème, non ?

Outre le fait qu'il peut insérer n'importe quel code HTML (et rendre votre page invalide), ce qui n'est pas le plus grave, il peut aussi ouvrir des balises de type `<script>` pour faire exécuter du code JavaScript au visiteur qui visualisera la page !

html+php
```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <script type="text/javascript">alert('Badaboum')</script> !</p>
```

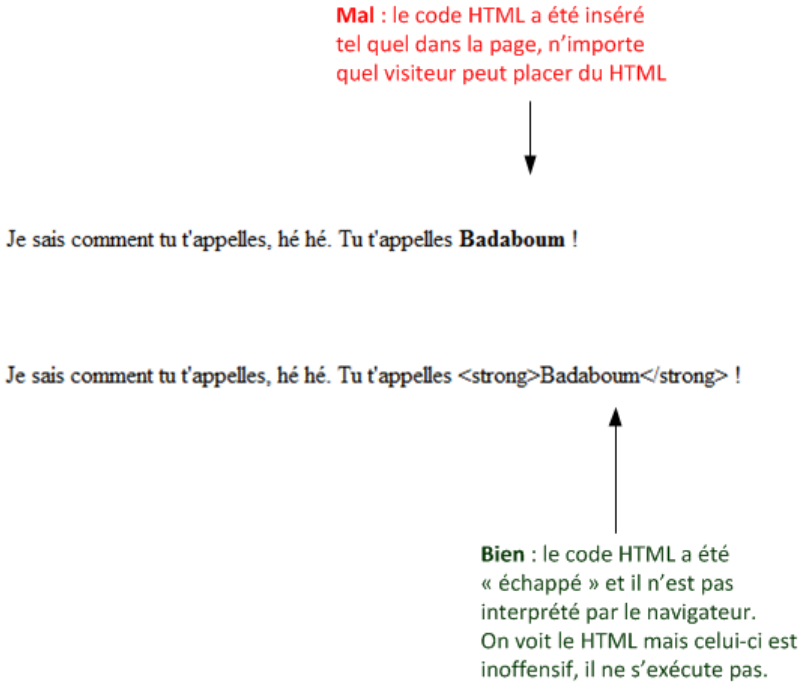
Tous les visiteurs qui arriveront sur cette page verront une boîte de dialogue JavaScript s'afficher. Plutôt gênant. Voyez la figure suivante.



Exécution de JavaScript par la faille XSS

Les choses deviennent vraiment critiques si le visiteur est assez malin pour récupérer vos cookies de cette façon-là. Les cookies stockent des informations sur votre session et parfois des informations plus confidentielles, comme votre pseudonyme et votre mot de passe sur le site ! Il est possible de forcer le visiteur qui lira le code JavaScript à envoyer tous les cookies qu'il a enregistrés pour votre site web, ce qui peut conduire au vol de son compte sur ce site.
Je ne rentrerai pas dans le détail de cette méthode car on s'éloignerait un peu trop du sujet, mais sachez que c'est possible et qu'il faut donc à tout prix éviter qu'un visiteur puisse injecter du code JavaScript dans vos pages.

Résoudre le problème est facile : il faut protéger le code HTML en l'échappant, c'est-à-dire en affichant les balises (ou en les retirant) plutôt que de les faire exécuter par le navigateur, comme sur la figure suivante.



Éviter la faille XSS en échappant le HTML

Pour échapper le code HTML, il suffit d'utiliser la fonction `htmlspecialchars` qui va transformer les chevrons des balises HTML `<>` en `<` et `>` respectivement. Cela provoquera l'affichage de la balise plutôt que son exécution.

html+php
```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <?php echo htmlspecialchars($_POST['prenom']); ?> !</p>
```

Le code HTML qui en résultera sera propre et protégé car les balises HTML insérées par le visiteur auront été échappées :

html+php
```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles &lt;strong&gt;Badaboum&lt;/strong&gt; !</p>
```

Il faut penser à utiliser cette fonction sur tous les textes envoyés par l'utilisateur qui sont susceptibles d'être affichés sur une page web. Sur un forum par exemple, il faut penser à échapper les messages postés par vos membres, mais aussi leurs pseudos (ils peuvent s'amuser à y mettre du HTML !) ainsi que leurs signatures. Bref, tout ce qui est affiché et qui vient à la base d'un visiteur, vous devez penser à le protéger avec `htmlspecialchars`.

Si vous préférez retirer les balises HTML que le visiteur a tenté d'envoyer plutôt que de les afficher, utilisez la fonction `strip_tags`.

Les éléments du formulaire L'envoi de fichiers

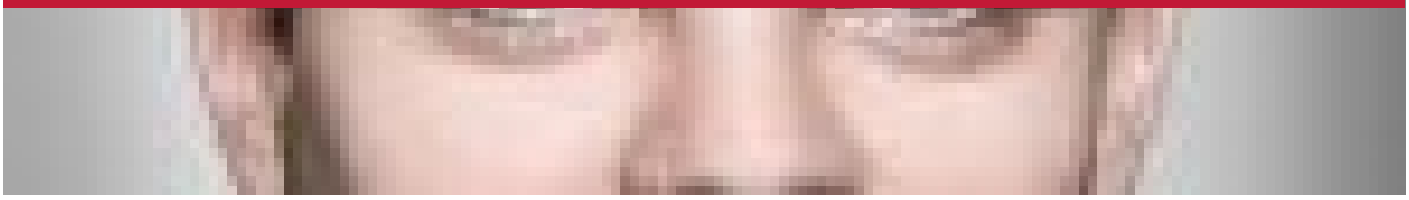
Partager

J'ai une question ou une remarque

Licence

L'auteur

Mathieu Nebra



OpenClassrooms

4001

Google + • Twitter

En ce moment sur OpenClassrooms

2 800 visiteurs en ligne

2 visiteurs sur cette page

7 463 779 messages sur le forum

Restez connecté à OpenClassrooms

Entrez votre e-mail OK

Conditions Générales d'Utilisation Roadmap Nous recrutons Qui sommes-nous ? Publicité Blog